

# Scalable zkSNARKs for Matrix Computations

## A Generic Framework for Verifiable Deep Learning

Mingshu Cong<sup>1</sup>, Sherman S.M. Chow<sup>2</sup>, Siu Ming Yiu<sup>1</sup>, Tsz Hon Yuen<sup>3</sup>

<sup>1</sup>The University of Hong Kong

<sup>2</sup>Chinese University of Hong Kong

<sup>3</sup>Monash University

Asiacrypt Presentation 2025.12



# Succinct zkML: Enabling AI-Empowered Blockchain

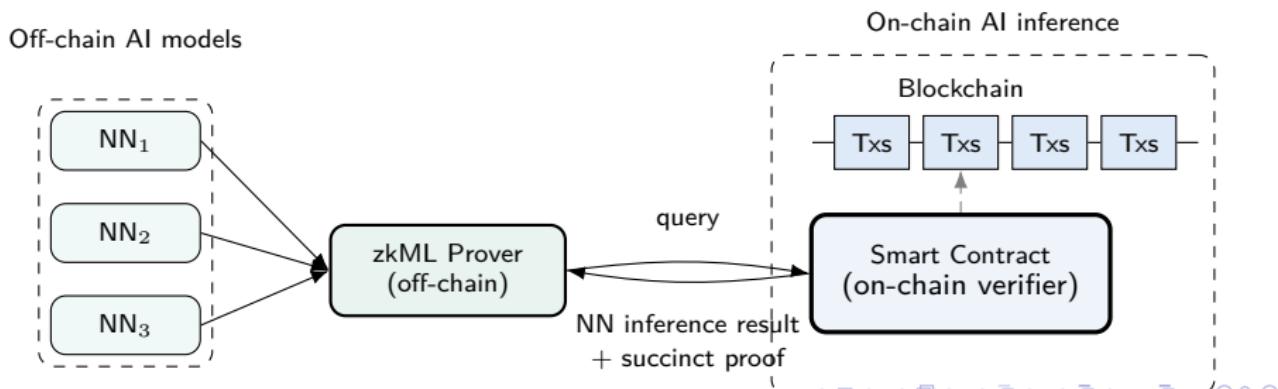
## Smart Contracts with Embedded AI Models

Future blockchain systems will require smart contracts that can:

- Enforce AML/KYC compliance on-chain
- Execute AI agent policies on-chain
- Perform autonomous risk assessment and monitoring

**Succinct zkML: off-chain inference & proof generation + on-chain verification**

Heavy ML computation is done by a single prover; all nodes verify a **succinct** proof.



# The Problem We Address

## Succinct Zero-Knowledge Machine Learning (zkML)

zkSNARKs (zero-knowledge **Succinct** Non-Interactive Arguments of Knowledge) for machine learning models (e.g., neural networks, NNs):  $Y = f(X)$ , where both the NN  $f$  and the input data  $X$  may be **secret**, but the inference result  $Y$  is publicly verifiable.

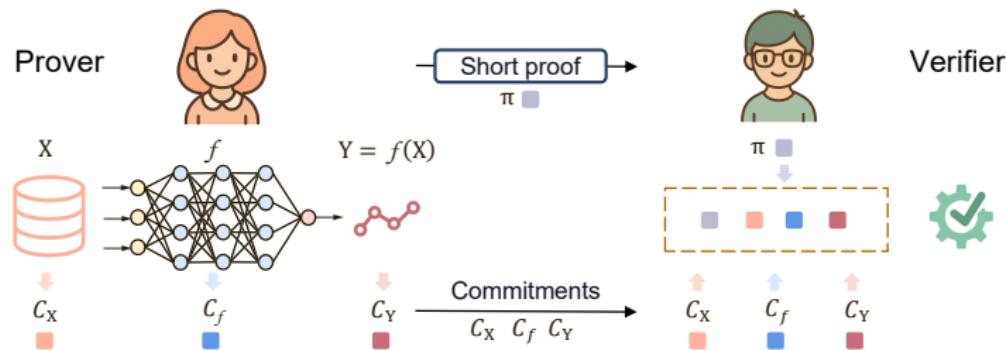


Figure: The prover produces a **short proof** to convince the verifier NN inference is correct.

# Neural Networks as Matrix Expressions

## A Generic View for zkML

NN computations can be represented as a directed acyclic graph (DAG) of matrix operations. If we design succinct proofs for DAG-structured matrix expressions, we obtain a generic zkML framework.

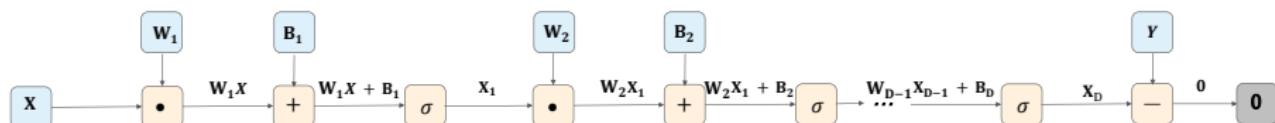


Figure: NN inference computation decomposes into a DAG of matrix operations.

- Fully connected layer:  $\mathbf{Y} = \sigma(\mathbf{W}\mathbf{X} + \mathbf{b})$
- Convolutional layer:  $\mathbf{Y} = \sigma(\text{conv}(\mathbf{W}, \mathbf{X}) + \mathbf{b})$
- Attention layer:  $\mathbf{Y} = \sigma\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}$

# Challenges of Succinct zkML

## Efficiency

We aim to achieve all of the following:

- **Linear prover time**
- **Logarithmic** proof size and verifier time
- **Practical** RAM usage for **billion-parameter** NNs

## NN Privacy

Hiding datasets and parameters via commitments is standard.

Hiding the **NN architecture** (the DAG topology) is much harder.

## Flexibility

A unified framework that supports **heterogeneous** NN structures and matrix operations across diverse NN designs.

# Building Block: LiteBullet via Folding

## New Inner Product Proof

LiteBullet is a Bulletproof-style protocol for proving  $c = \vec{a} \bullet \vec{b}$  using **only field operations**, without converting vectors into polynomials (**polynomial-free**).

- **Folding step.** Split vectors into left/right halves:

$\vec{a} = (\vec{a}_L, \vec{a}_R)$ ,  $\vec{b} = (\vec{b}_L, \vec{b}_R)$ . For a random challenge  $x$ :

$$(\vec{a}_L + x^{-1}\vec{a}_R) \bullet (\vec{b}_L + x\vec{b}_R) = c + x^{-1}(\vec{a}_L \bullet \vec{b}_R) + x(\vec{a}_R \bullet \vec{b}_L).$$

- **Effect of folding.** One folding step reduces an  $n$ -dimensional check to a  $(n/2)$ -dimensional one.
- **Recursive reduction.** After  $\log_2 n$  rounds,  $c = \vec{a} \bullet \vec{b}$  reduces to a scalar product:

$$\hat{c} = \hat{a} \cdot \hat{b},$$

where  $\hat{a} = \vec{a} \bullet \vec{\xi}^{-1}$ ,  $\hat{b} = \vec{b} \bullet \vec{\xi}$  for tensor-structured random vectors  $\vec{\xi}^{-1}$ ,  $\vec{\xi}$  from the verifier's challenges.

- **Final step.** The prover employs additional protocols to prove:

$$\hat{a} = \vec{a} \bullet \vec{\xi}^{-1}, \quad \hat{b} = \vec{b} \bullet \vec{\xi}, \quad \text{for public tensor-structured } \vec{\xi}^{-1}, \vec{\xi}.$$

# LiteBullet Applied to Matrix Multiplication

## Freivalds' Algorithm (Review)

To check whether  $ab = c$ , Freivalds' algorithm verifies:

$$ab \vec{r} = c \vec{r},$$

for a random vector  $\vec{r} \in \mathbb{F}^n$ .

- **Two-sided projection.** We apply Freivalds' Algorithm twice and prove:

$$\vec{l}^\top ab \vec{r} = \vec{l}^\top c \vec{r},$$

where  $\vec{l} \in \mathbb{F}^m$  and  $\vec{r} \in \mathbb{F}^n$  are random vectors.

- **Reduction via LiteBullet.** Define:

$$v := \vec{l}^\top c \vec{r} = \vec{l}^\top ab \vec{r} = (\vec{l}^\top a) \bullet (b \vec{r}).$$

Applying LiteBullet to this inner product reduces to checking:

$$\hat{a} = \vec{l}^\top a \vec{\xi}^{-1}, \quad \hat{b} = \vec{\xi}^\top b \vec{r},$$

where  $\vec{\xi}^{-1}$  and  $\vec{\xi}$  are public tensor-structured vectors.

# Reducing Parent Projection to Child Projections

- **Structured randomness.** Use tensor-structured vectors  $\vec{l}$  and  $\vec{r}$  derived from the random challenges  $x_{l,1}, \dots, x_{l,\log_2 m}$  and  $x_{r,1}, \dots, x_{r,\log_2 n}$ :

$$\vec{l} = (1, x_{l,1}) \otimes \cdots \otimes (1, x_{l,\log_2 m}), \quad \vec{r} = (1, x_{r,1}) \otimes \cdots \otimes (1, x_{r,\log_2 n}).$$

- **Interpretation.** Using LiteBullet, proving  $v = \vec{l}^\top c \vec{r}$  reduces to proving:

$$\hat{a} = \vec{l}^\top a \vec{\xi}^{-1} \quad \text{and} \quad \hat{b} = \vec{\xi}^\top b \vec{r}.$$

$v$ : projection of the output matrix  $c$  onto  $(\vec{l}, \vec{r})$ .

$\hat{a}$ : projection of the left input matrix  $a$  onto  $(\vec{l}, \vec{\xi}^{-1})$ .

$\hat{b}$ : projection of the right input matrix  $b$  onto  $(\vec{\xi}, \vec{r})$ .

- **DAG View.** In the DAG of matrix expressions, proving a *parent* projection reduces to proving its *child* projections.

# Proof Reduction for Various Matrix Operations

For all **Type-1 operations**  $\circledast$ , include matrix addition, multiplication, transpose, inner product, Hadamard product, Kronecker product, concatenation, and truncation, Evalyn applies the same reduction principle.

## Atomic Evalyn Proof

Given random vectors  $\vec{l}$  and  $\vec{r}$ , proving  $v = \vec{l}^\top (\mathbf{a} \circledast \mathbf{b}) \vec{r}$  reduces to proving:

$$v_L = \vec{l}_L^\top \mathbf{a} \vec{r}_L, \quad v_R = \vec{l}_R^\top \mathbf{b} \vec{r}_R,$$

where  $(\vec{l}_L, \vec{r}_L, \vec{l}_R, \vec{r}_R)$  are tensor-structured vectors related to  $(\vec{l}, \vec{r})$ .

$$\left\{ \begin{array}{l} \mathcal{P} \text{ produces a } \circledast\text{-specific proof } \pi \text{ for } v = \vec{l}^\top (\mathbf{a} \circledast \mathbf{b}) \vec{r} \\ \mathcal{V} \text{ checks } \pi \text{ and:} \\ \{v_L = \vec{l}_L^\top \mathbf{a} \vec{r}_L \wedge v_R = \vec{l}_R^\top \mathbf{b} \vec{r}_R\} \text{ (recursively proven), where:} \\ v_L \text{ and } v_R \text{ are the final elements of } \pi, \text{ and} \\ \vec{l}_L, \vec{r}_L, \vec{l}_R, \vec{r}_R \text{ are derived from } \vec{l}, \vec{r}, \text{ and the randomness from } \mathcal{V} \end{array} \right\} .$$

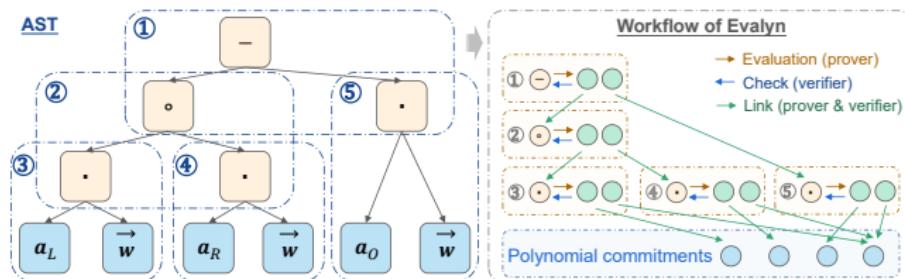
# Backward Proof Reduction

## Post-Order Traversal for Node Evaluation

First, traverse the DAG in **post-order** to evaluate all nodes from leaves to the root.

## Pre-Order Traversal for Proof Generation

Then, traverse the DAG in **pre-order** to recursively reduce the root-node projection to leaf-node projections.



**Figure:** DAG for  $\vec{0} = (a_L \vec{w}) \circ (a_R \vec{w}) - a_O \vec{w}$  and the workflow for generating zkSNARKs through DAG traversal.

# From Type-2 Operations to Type-1 Operations

## Type-1 Operations

Operations that admit an efficient projection reduction proof (e.g., matrix multiplication, addition, etc.).

## Type-2 Operations

Operations for which no such efficient projection reduction proof is known (e.g., nonlinear activations, convolution, etc.).

## Our Strategy

Express Type-2 operations as **equivalent Type-1 matrix expressions**, so they can be handled by a **unified reduction framework**.

## Example: Lookup Proof for Nonlinear Activation

- Nonlinear activations are Type-2 operations.
- For a monotone increasing  $\sigma$ , each output  $c = \sigma(a)$  corresponds to an interval of valid preimages. The vectors  $\mathbf{k}_1$  and  $\mathbf{k}_2$  store the minimum and maximum preimages for every possible function value  $c$ . The lookup table is:

$$(\mathbf{k}_1, \mathbf{k}_2, \mathbf{v}) = \{(a_1, a_2, c) : c = \sigma(a_1) = \sigma(a_2), a_1 \leq b \leq a_2 \quad \forall b \in \sigma^{-1}(c)\}.$$

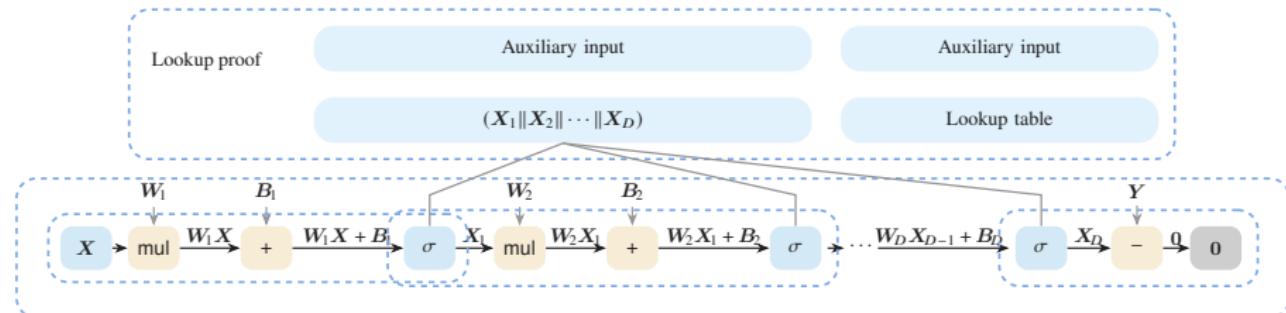
- To prove  $c = \sigma(a)$ , it suffices to show  $a_1$  and  $a_2$  such that:

$$a_1 \leq a \leq a_2, \quad \{(a_1, a_2, c)\} \in \{(\mathbf{k}_1, \mathbf{k}_2, \mathbf{v})\}.$$

- Lasso [6] provides a lookup proof, which we express as vector equations to fit into our framework.

# Committing to New Leaf Nodes

Transforming Type-2 operations into Type-1 expressions introduces **new leaf nodes** into the DAG. These nodes must be **committed** before applying proof reduction.



**Figure:** Commitments to new leaf nodes (blue rectangles) resulting from the Type-2 to Type-1 transformation.

# Succinctness via PoP

## Non-Succinct Proof from Proof Reduction

- Each DAG node yields at least a constant-size proof, so the total proof size grows with the number of nodes and breaks succinctness.
- For a network with  $D$  layers of  $n \times n$  matrices, the proof size is  $O(D \log n)$  —not succinct.

## Proof of Proof (PoP)

We use an additional zkSNARK that proves the (non-succinct) verifier outputs true on the original reduction proof.

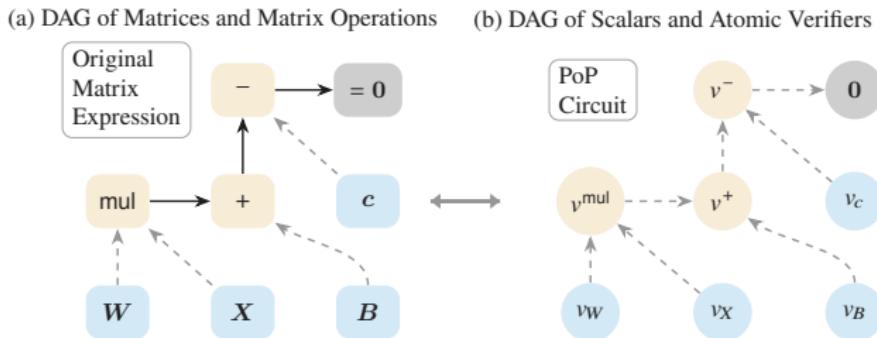
- This compresses the proof size and verifier time to  $O(\log(Dn))$ .
- It also enables hiding the NN structure by committing to the PoP circuit.

# Hiding NN Architecture via PoP Commitments

## Key Insight

The PoP circuit preserves the topology of the original matrix DAG.

- Represent the NN as a DAG of atomic matrix operations.
- Traverse the DAG to generate a sublinear proof. (**Proof reduction**)
- Commit to a **smaller PoP circuit** to hide the NN architecture.



**Figure:** DAG for  $c = WX + B$  and its PoP circuit. Each matrix is projected to a scalar, yielding a scalar DAG with the same topology, which becomes the PoP circuit.  
(Legend: root, leaf, operation)

# Workflow of Evalyn

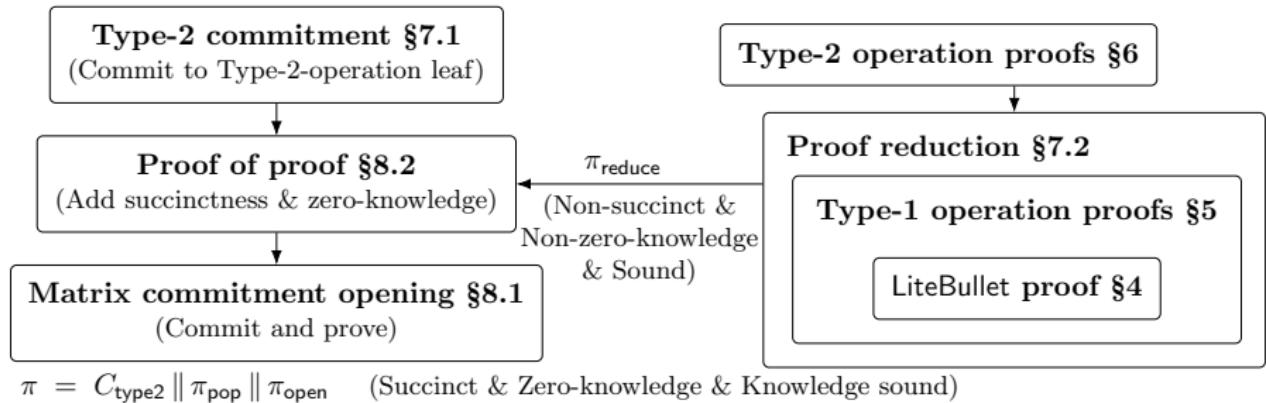
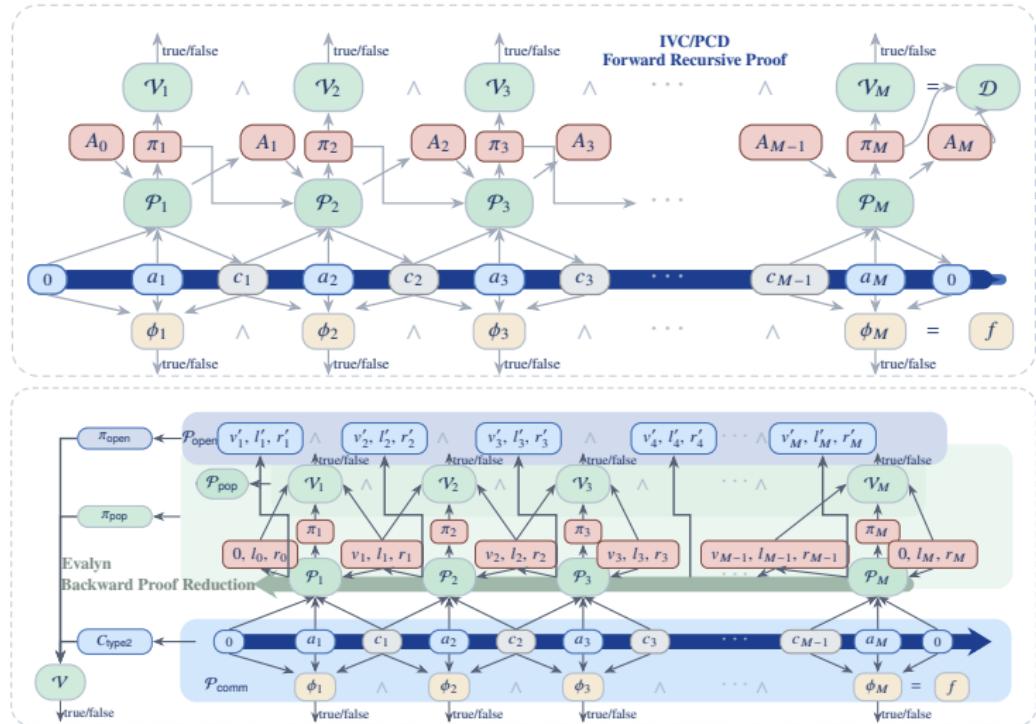


Figure: The Evalyn framework.

## Comparison with IVC/PCD



**Figure:** Comparison with IVC/PCD.

# Asymptotic Efficiency

Table: Architecture/operation-specific proofs ( $D$ -layer NN inference,  $n \times n$  matrices)

Protocol	Arch. <sup>a</sup>	Op. <sup>a, b</sup>	Prover Time	Proof Size	Verifier Time	Hiding <sup>a</sup>
Zen [1]	MLP/CNN	Conv.	$O(Dn^4 \log n)^c$	$O(1)$	$O(1)$	Arch.&Par.
pvCNN [8]	CNN	Conv.	$O(Dn^4 \log n)^c$	$O(1)$	$O(1)$	Arch.&Par.
vCNN [4]	CNN	Conv.	$O(Dn^2 \log n)$	$O(D)$	$O(D)$	Par.
KMC18 [3]	MLP	Basics	$O(Dn^2 \log n)$	$O(D \log n)$	$O(D \log n)$	Par.
Trustless [2]	MLP	Basics	$O(Dn^2)$	$O(D \log n)$	$O(D \log n)$	Par.
zkCNN [5]	CNN	Conv.	$O(Dn^2)$	$O(D \log^2 n)$	$O(Dn)^d$	Par.
zkLLM [7]	LLM	Softmax	$O(Dn^2)$	$O(D \log n)$	$O(Dn)^d$	Par.
Evalyn	General	General	$O(Dn^2)$	$O(\log(Dn))$	$O(\log(Dn))$	Arch.&Par.

<sup>a</sup> Shorthand: Arch.: Architecture; Op.: Operation; Par.: Parameter.

<sup>b</sup> Matrix multiplication and ReLU are the basics and are omitted. Conv.: Convolution.

<sup>c</sup> Complexities for  $n \times n$  matrix multiplication/convolution itself are  $O(n^3)/O(n^4)$ .

<sup>d</sup> zkCNN/zkLLM use Hyrax PCS; verifier time complexity larger than its proof size.

# Concrete Efficiency

We benchmark an MLP with  $2^{10}$  layers, each using a  $2^{10} \times 2^{10}$  weight matrix ( $\sim 10^9$  parameters total). The prover times were 34.8 min/36.8 min.

Table: Performance on a  $2^{10}$ -layer NN ( $\sim 1$  billion parameters)

Layer $i$ : $\mathbf{X}_i \leftarrow \text{sigmoid}(\mathbf{W}_i \mathbf{X}_{i-1} + \mathbf{B}_i)$	8-Bit	16-Bit
Unverified NN inference (no proof)	3.53 s	4.64 s
Setup time	2.44 s	2.43 s
SRS size	22.02 MB	22.02 MB
NN commitment time	2,023.87 s	2,097.80 s
Structure commitment (single thread)	747.07 s	737.30 s
Parameter commitment	1,276.80 s	1,360.50 s
NN commitment size	42.27 KB	42.27 KB
Prover time	2,090.71 s	2,210.88 s
Auxiliary input commitment	733.08 s	755.61 s
Proof reduction ( $\mathbb{F}$ operations)	218.80 s	293.16 s
PoP proving (single thread)	775.36 s	765.05 s
Leaf node opening	25.46 s	31.23 s
Fiat–Shamir transform proving	338.01 s	356.83 s
Proof size	98.84 KB	98.84 KB
Verifier time (single thread)	173.19 ms	172.72 ms
Peak RAM	87.42 GB	103.63 GB

Parallelized across 64 CPU threads unless noted “single thread”

# Summary of Contributions

- A generic and **succinct** zkML framework
- Hides both NN parameters and **NN structures**
- Competitive **efficiency** with practical prover time
- **Flexibility** across diverse NN structures and operations
- A **polynomial-free** inner product proof (LiteBullet)

## References

- [1] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: An optimizing compiler for verifiable, zero-knowledge neural network inferences. *IACR Cryptol. ePrint Arch.* 2021/087, 2021.
- [2] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up trustless DNN inference with zero-knowledge proofs. *arXiv:2210.08674*, 2022.
- [3] Julien Keuffer, Refik Molva, and Hervé Chabanne. Efficient proof composition for verifiable computation. In *ESORICS Part I*, pages 152–171, 2018.
- [4] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network based on zk-SNARKs. *IEEE TDSC*, 21(4):4254–4270, 2023.
- [5] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *CCS*, pages 2968–2985, 2021.
- [6] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In *EUROCRYPT Part VI*, pages 180–209, 2024.
- [7] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *CCS*, pages 4405–4419, 2024.
- [8] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. pvCNN: Privacy-preserving and verifiable convolutional neural network testing. *IEEE Trans. Inf. Forensics Secur.*, 18:2218–2233, 2023.

Thank you!  
Questions are welcome.